A data aggregation protocol based on TFHE

Maria Ferrara, Antonio Tortora and Maria Tota*

Dipartimento di Matematica e Fisica, Università della Campania "Luigi Vanvitelli"
viale Lincoln, 5 - 81100 - Caserta, Italy
E-mail: maria.ferrara1@unicampania.it, antonio.tortora@unicampania.it

Dipartimento di Matematica, Università di Salerno
via Giovanni Paolo II, 132 - 84084 - Fisciano (SA), Italy
E-mail: mtota@unisa.it

Abstract

Torus Fully Homomorphic Encryption (TFHE) is a probabilistic cryptosytem over the real torus which allows one to operate directly on encrypted data without first decrypting them. We present an aggregation protocol based on a variant of TFHE for computing the sum of sensitive data, working only with the corresponding ciphertexts. Our scheme is an ideal choice for a system of smart meters - electronic devices for measuring energy consumption - that demands consumers' privacy. In contrast to some other solutions, our proposal does not require any communication among smart meters and it is quantum-safe.

Keywords: TFHE, fully homomorphic encryption, data aggregation, privacy
2020 Mathematics Subject Classification: 94A60, 08A99

1 Introduction

In the last few years, much attention has been devoted to the aggregation of time-series data from smart meters, which are electronic devices that record

*The authors are members of *National Group for Algebraic and Geometric Structures, and their Applications* (GNSAGA–INdAM), and members of the non-profit association *Advances in Group Theory and Applications.*

the electric energy consumption, at regular intervals of 30 minutes or less, and communicate it to an energy supplier. The use of smart meters can make any energy distribution system more efficient for the energy supplier and final customers, by monitoring the effective amount of energy used: the former can develop an energy plan to prevent unnecessary energy production and the latter can take advantage of different prices and more accurate billing. However, the collected data may contain sensitive consumer information, and therefore a privacy-preserving solution is needed.

In literature there are several proposals for privacy protection in smart metering (see [9] or [12] for a survey). They can be classified according to three main techniques such as *anonymization*, *perturbation*, and *aggregation*. In particular, for the aggregation, data can be aggregated by a trusted third party or by making use of the homomorphic property of some cryptosystems. For example, in [3], the aggregation is performed by using an additive homomorphic version of ElGamal cryptosystem (see [6]), where a discrete logarithm computation is necessary for obtaining the desired information, say μ. Hence, the computation can be done efficiently when μ is not too large. Other solutions are based on the Paillier cryptosystem [14] and its additive homomorphic property. Nevertheless, it is well-known that the ElGamal and Paillier cryptosystems are not quantum-safe.

In this paper, we present a protocol that enables an unreliable data aggregator to compute the sum of several plaintexts (readings in the case of smart meters), working only with the corresponding ciphertexts. More precisely, we assume to have l devices and that each of them sends an encrypted plaintext $e(\mu_i)$ to the aggregator, that obtains $\mu = \mu_1 + \ldots + \mu_l$ without knowing any μ_i. Our approach is similar to the energy monitoring system proposed in [3], even if we require a trusted third party for the generation of a group key for all devices. On the other hand, our protocol bases its safety on the Learning With Errors problem that is widely considered to be post-quantum.

1.1 Fully homomorphic encryption

For our protocol, we will rely on a Fully Homomorphic Encryption (FHE) scheme, that is, a probabilistic cryptosystem that allows one to operate directly on encrypted data without first decrypting them. Given a non-empty finite set $\mathcal{S}$, a *probabilistic cryptosystem* may be seen as a five-tuple $(\mathcal{P} \times \mathcal{S}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ of finite sets, where $\mathcal{P}, \mathcal{C}$ and $\mathcal{K}$ are respectively the sets of plaintexts, ciphertexts and keys, whereas $\mathcal{E}$ and $\mathcal{D}$ are the following sets:

$$\mathcal{E} = \left\{ e_k : \mathcal{P} \times \mathcal{S} \to \mathcal{C} \,|\, k \in \mathcal{K} \right\}, \quad \mathcal{D} = \left\{ d_k : \mathcal{C} \to \mathcal{P} \,|\, k \in \mathcal{K} \right\};$$

moreover we require that, for any encryption function $e_{k_1} \in \mathcal{E}$, there is a decryption function $d_{k_2} \in \mathcal{D}$ such that $d_{k_2}\big(e_{k_1}(x,r)\big) = x$ for all $x \in \mathcal{P}$ and $r \in \mathcal{S}$. Note that, unlike a deterministic cryptosystem, a random value $r \in \mathcal{S}$ is used to encrypt x and so any plaintext can be encrypted in different ways. Following [8], we say:

Definition 1.1. A probabilistic cryptosystem is *somewhat homomorphic* if $\mathcal{P}$ and $\mathcal{C}$ are each equipped with two operations, say $+, \cdot$ for $\mathcal{P}$ and $\oplus, *$ for $\mathcal{C}$, and

1. $(\mathcal{P}, +, \cdot)$ and $(\mathcal{C}, \oplus, *)$ are rings,
2. there exists $\mathcal{U} \subseteq \mathcal{S}$, with $\mathcal{U} \neq \emptyset$, such that, for any $x_i, y_i \in \mathcal{P}$ and any $u_i, v_i \in \mathcal{U}$, there exist $t, z \in \mathcal{S}$ for which

$$e_{k_1}(x_1, u_1) \oplus e_{k_1}(x_2, u_2) = e_{k_1}(x_1 + x_2, t),$$
$$e_{k_1}(y_1, v_1) * e_{k_1}(y_2, v_2) = e_{k_1}(y_1 \cdot y_2, z),$$

whenever $e_{k_1} \in \mathcal{E}$. In addition, the cryptosystem is *fully homomorphic* if the elements t and z belong to $\mathcal{U}$.

As a consequence, for a somewhat homomorphic cryptosystem, we could do other additions or multiplications: for example, $e_{k_1}(x_1 + x_2, t) \oplus e_{k_1}(x_2, u_2)$ if $t \in \mathcal{U}$, or $e_{k_1}(y_1 \cdot y_2, z) * e_{k_1}(y_2, v_2)$ if $z \in \mathcal{U}$. On the other hand, for a fully homomorphic cryptosystem where $\mathcal{P} = \mathcal{C}$, and for any function

$$f : \underbrace{\mathcal{P} \times \ldots \times \mathcal{P}}_{m} \to \mathcal{P}$$

such that $f(x_1, \ldots, x_m)$ is given by some additions and multiplications of the x_i's in the ring $\mathcal{P}$, we have

$$d_{k_2}\big(f\big(e_{k_1}(x_1, u_1), \ldots, e_{k_1}(x_m, u_m)\big)\big) = f(x_1, \ldots, x_m)$$

for all $x_i \in \mathcal{P}$ and $u_i \in \mathcal{U}$.

1.2 The bootstrapping

In 2009, Gentry put forward the first fully homomorphic encryption scheme [11] (see also [10]). His idea consists in taking a somewhat homomorphic cryptosystem and make it fully homomorphic. For this purpose, it is needed to produce a new ciphertext before proceeding with further operations. Indeed, the security of communications in a somewhat homomorphic cryptosystem

is ensured by the presence of some noise in ciphertexts; however, the noise level increases during homomorphic operations until when it is impossible to decrypt. To restrict the growth of noise, Gentry used a groundbreaking technique, the so-called *bootstrapping*, that can be described as follows.

Consider a somewhat homomorphic cryptosystem, as in Definition 1.1. Suppose futher that, for a given plaintext x, the element $(x,t) \in \mathcal{P} \times \mathcal{S}$ is *noisy*, in the sense that $t \notin \mathcal{U}$. Then, for a function $g_k : \mathcal{C} \to \mathcal{C}$, depending on the (public) *bootstrapping key* $k = e'_{k_3}(k_1, r)$ where e'_{k_3} is an encryption function (not necessarily in $\mathcal{E}$), the bootstrapping enables to reduce the noise: indeed, it yields an element $u \in \mathcal{U}$ such that $g_k\big(e_{k_1}(x,t)\big) = e_{k_1}(x,u)$. Similarly, if $y \in \mathcal{P}$ and $z \notin \mathcal{U}$, we get $g_k\big(e_{k_1}(y,z)\big) = e_{k_1}(y,v)$ for some $v \in \mathcal{U}$. Hence, one can compute $e_{k_1}(x_i,u_i) \oplus e_{k_1}(x,u)$ and $e_{k_1}(y_i,v_i) * e_{k_1}(y,v)$.

Following Gentry's scheme, many other FHE cryptosystems were introduced but all these had a common issue: the bootstrapping took too long to run (minutes) and the bootstrapping key was too large (gigabytes). In 2015, Ducas and Micciancio [7] presented a new method to reduce the running time of the bootstrapping procedure. Their work inspired the birth of the Torus Fully Homomorphic Encryption (TFHE) cryptosystem [4], which runs the bootstrapping in terms of milliseconds and produces a bootstrapping key whose size is measured in megabytes rather than gigabytes.

TFHE will be explained in detail in Section 2 and then will be used, in Section 3, for the above-mentioned aggregation protocol.

2 Torus fully homomorphic encryption

The letter T in TFHE stands for the real *torus* $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. TFHE is implemented in the open-source library `CONCRETE` [5] where, for technical requirements, the elements $0, \frac{1}{q}, \ldots, \frac{q-1}{q}$ of $\mathbb{T}$ are identified with the elements of $\mathbb{Z}_q$, i.e., the group of integers modulo $q = 2^r$ with $r = 32$ or 64.

This variant of TFHE can be described as follows [8].

Definition 2.1. Put $\mathbb{B} = \{0,1\}$ (or $\mathbb{Z}_q$) and let h, n be positive integers with $h < r - 1$.

1. The plaintext space of TFHE is
$$\mathcal{P} = \big\{\mu \in \mathbb{Z}_q \,|\, \mu = \mu_h \cdot 2^h + \ldots + \mu_{r-1} \cdot 2^{r-1}, \mu_i \in \mathbb{B}\big\},$$
whereas $\mathcal{C} = \mathbb{Z}_q^{n+1}$ is the ciphertext space.

2. For a secret key $s = (s_1, \ldots, s_n) \in \mathbb{B}^n$ and a *random mask* $a = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$, the encryption function is given by
$$E_s : (\mu, a, e) \in \mathcal{P} \times \mathcal{S} \mapsto (a, b) \in \mathcal{C}$$

where $\mathcal{S} = \mathbb{Z}_q^n \times \mathcal{U}$ with

$$\mathcal{U} = \big\{ e \in \mathbb{Z}_q \,|\, e = \epsilon_0 \cdot 2^0 + \ldots + \epsilon_{h-1} \cdot 2^{h-1}, \epsilon_i \in \mathbb{B} \big\},$$

and

$$b = \sum_{k=1}^{n} a_k s_k + \mu + e \pmod{q}.$$

3. The decryption function is $\pi \circ \varphi$, indeed the composition of functions

$$\varphi : (a, b) \in \mathcal{C} \mapsto b - \sum_{k=1}^{n} s_k a_k \in \mathbb{Z}_q$$

and

$$\pi : \mu_0 \cdot 2^0 + \ldots + \mu_{r-1} \cdot 2^{r-1} \in \mathbb{Z}_q \mapsto \mu_h \cdot 2^h + \ldots + \mu_{r-1} 2^{r-1} \in \mathcal{P};$$

hence $\pi\big(\varphi(a, b)\big) = \pi(\mu + e) = \mu$.

Although $\mathcal{P}$ is only a subgroup of the additive group of $\mathbb{Z}_q$, TFHE is somewhat homomorphic with respect to the addition. In fact, for $\mu_1, \mu_2 \in \mathcal{P}$ and $a^{(1)} = (a_{11}, \ldots, a_{1n}), a^{(2)} = (a_{21}, \ldots, a_{2n}) \in \mathbb{Z}_q^n$, we have

$$E_s\big(\mu_1, a^{(1)}, e_1\big) + E_s\big(\mu_2, a^{(2)}, e_2\big) = E_s\big(\mu_1 + \mu_2, a^{(1)} + a^{(2)}, e_1 + e_2\big),$$

provided that $e_1 + e_2 \in \mathcal{U}$. For example, in order to compute

$$E_s\big(\mu_1 + \mu_2, a^{(1)} + a^{(2)}, e_1 + e_2\big) + E_s\big(\mu_3, a^{(3)}, e_3\big),$$

the bootstrapping is needed when

$$\mathcal{U} = \big\{ e \in \mathbb{Z}_q \,|\, e = \epsilon_0 \cdot 2^0 + \ldots + \epsilon_{h-2} \cdot 2^{h-2}, \epsilon_i \in \mathbb{B} \big\},$$

for $h > 1$, $e_i \in \mathcal{U}$ and $e_1 + e_2 \notin \mathcal{U}$: for $e_1 + e_2 + e_3$ could not belong to $\mathcal{U}$. The result is a new ciphertext of $\mu_1 + \mu_2$, that is $E_s\big(\mu_1 + \mu_2, a^{(1)} + a^{(2)}, e_1 + e_2\big) = E_s\big(\mu_1 + \mu_2, a, e\big)$ for some a and e, with $e \in \mathcal{U}$.

Suppose now that $b = \sum_{i=1}^{n} s_i a_i + \mu^*$, with $\mu^* = \mu + e$, is a noisy LWE ciphertext. Let consider the so-called *test polynomial*

$$t(x) = t_0 + t_1 x + \ldots + t_{N-1} x^{N-1} \in \mathbb{Z}_q[x]/I$$

where $N > 1$ is a power of 2, I is the ideal generated by the polynomial $x^N + 1$ and each $t_j = \pi(\frac{q}{2N} j)$ with π as in Definition 2.1.3. Assume further

that b has at least one bit of padding left, that is $b < 2^{r-1}$. Then $\mu^* < 2^{r-1}$ and it is easy to see that $\bar{\mu}^* = \lfloor \frac{2N}{q}\mu^* \rceil \leq N-1$. Hence, $t_{\bar{\mu}^*} = \mu$.

In TFHE the bootstrapping procedure involves three main algorithms: the blind rotation, the sample extraction and the key switching. The blind rotation consists in finding a Ring-LWE ciphertext of the polynomial $x^{-\bar{\mu}^*}t(x)$, whose constant term is actually $t_{\bar{\mu}^*}$. This constant term is then extracted, by using the sample extraction, as a refreshed LWE encryption of μ with less noise, but under a different key. Finally, the key switching algorithm converts the new LWE ciphertext into a LWE ciphertext under the initial key s.

For completeness, we mention that the above bootstrapping is also *programmable*, in the sense that it enables the evaluation of a given function on the input ciphertext while reducing the noise. This makes TFHE a fully homomorphic cryptosystem over real numbers: in fact, the product $x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$ of two real numbers can be computed using two bootstrapping operations with the real function $z \mapsto \frac{z^2}{4}$. For more details we refer the reader to [5] or [8, Section 4].

2.1 Learning with errors

The security of TFHE is based on the Learning With Errors (LWE) problem, introduced by Regev in [17] (see also [13] for its variant on rings). In our context, for $n \geq 1$ and $q \geq 2$, the (search-)LWE problem asks to recover $s = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$ given any desired $m = poly(n)$ independent linear equations such as the following

$$\begin{cases} a_{11}s_1 + \ldots + a_{1n}s_n + e_1 = b_1 \pmod q \\ a_{21}s_1 + \ldots + a_{2n}s_n + e_2 = b_2 \pmod q \\ \quad \vdots \qquad\qquad \vdots \qquad\quad \vdots \\ a_{m1}s_1 + \ldots + a_{mn}s_n + e_m = b_m \pmod q \end{cases}$$

where the matrix $A = (a_{jk}) \in \mathbb{Z}_q^{m \times n}$ is chosen uniformly, and each e_i is usually taken from a Gaussian distribution χ and rounded to the nearest integer (modulo q). Of course, when $e_1 = e_2 = \ldots = e_m$, after about $n+1$ equations, it is possible to recover s in polynomial time using Gaussian elimination. There is also a decision version of the LWE problem, that is as follows: given (A, b), with A as before, distinguish with some non-negligible probability if $b = (b_1, \ldots, b_m)$ is chosen uniformly from $\mathbb{Z}_q^m$ or chosen to be $As + e$ with $e = (e_1, \ldots, e_m) \in \mathbb{Z}_q^m$ taken according to χ. However, by [2, Theorem 2.15], the search-LWE and decision-LWE problems are equivalent whenever q is a power of 2.

The LWE problem is believed to be computationally hard, because of a reduction from the worst-case hardness of some lattice problems, such as GapSVP (the decision version of the Shortest Vector Problem), to the search-LWE problem [17] (see also [16]). Recall that the lattice L generated by some vectors $v_1, \ldots, v_n \in \mathbb{R}^m$ is the set of all linear combinations of $v_1, \ldots, v_n$ with coefficients in $\mathbb{Z}$. Furthermore, the shortest vector problem consists in finding a non-zero vector $v \in L$ that minimizes the Euclidean norm.

We also point out that the above Regev's reduction works for any modulus $q \geq 2\sqrt{n}/\alpha$, with $0 < \alpha < 1$, but it requires the use of quantum computation. In [15], Peikert proved that LWE is classically at least as hard as worst-case GapSVP on lattices for large modulus $q \geq 2^{n/2}$. Since then, the LWE problem has become one of the most attractive and promising topics for post-quantum cryptography.

2.2 TFHE: from private-key to public-key

TFHE is a private-key encryption scheme which, according to [18], can be converted into a public-key encryption scheme, as follows.

First note that, given the private-key $s = (s_1, \ldots, s_n) \in \mathbb{B}^n$, a plaintext $\mu \in \mathcal{P}$ and a mask $a = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$, we have

$$E_s(\mu, a, e) = E_s(0, a, e) + (\underbrace{0, \ldots, 0}_{n}, \mu) = (a_1, \ldots, a_n, \overline{b}) + (\underbrace{0, \ldots, 0}_{n}, \mu)$$

where $\overline{b} = \sum_{k=1}^{n} a_k s_k + e \pmod{q}$. Consider now m encryptions of 0, namely $c_j = (a_{j1}, \ldots, a_{jn}, b_j)$ with $b_j = \sum_{k=1}^{n} a_{jk} s_k + e_j$ for any $j \in \{1, \ldots, m\}$. Since TFHE is additively somewhat homomorphic, any linear combination of c_j's is also an encryption of 0, provided that the resulting noise $e_1 + \ldots + e_m \in \mathcal{U}$. This leads to a public-key version of TFHE, where the public key is the matrix

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} & b_1 \\ a_{21} & \ldots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & \ldots & a_{mn} & b_m \end{pmatrix} \in \mathbb{Z}_q^{m \times (n+1)}$$

and the encryption function is given by

$$E_A : (\mu, \beta_1, \ldots, \beta_m) \in \mathcal{P} \times \mathbb{B}^m \mapsto \begin{pmatrix} \beta_1 & \ldots & \beta_m \end{pmatrix} A + \begin{pmatrix} 0 & \overset{n}{\ldots} & 0 & \mu \end{pmatrix} \in \mathbb{Z}_q^{(n+1)}.$$

Hence, the encryption of μ is obtained by adding a random subset of encryptions of 0 to $(\underbrace{0, \ldots, 0}_{n}, \mu)$. On the other hand, if $E_A(\mu, \beta_1, \ldots, \beta_m) =$

$\begin{pmatrix} a'_1 & \dots & a'_n & b' \end{pmatrix}$, the plaintext μ can be recovered computing $\pi(b' - \sum_{k=1}^{n} a'_k s_k)$ with π as in Definition 2.1.3.

3 The protocol

Our protocol requires a trusted third party (TTP), a data aggregator, and $l > 2$ devices sharing their data with the aggregator. The data aggregation is preceded by a phase that involves the TTP in the generation of a group key for all devices.

3.1 Group key generation

In what follows we refer to the notation of Definition 2.1. A sketch of this first step is given in Fig. 1.

1. For a secret key $\overline{s} = (\overline{s}_1, \dots, \overline{s}_n) \in \{0,1\}^n$, the TTP chooses m encryptions of 0, say $(a_{j1}, \dots, a_{jn}, b_j)$ with $b_j = \sum_{k=1}^{n} a_{jk}\overline{s}_k + e_j$ for any $j \in \{1, \dots, m\}$, and sends the matrix

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ a_{21} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} & b_m \end{pmatrix}.$$

   to each device D_i, $i = 1, \dots, l$.

2. Each D_i, storing a secret key $(s_{i1}, \dots, s_{in}) \in \mathbb{Z}_q^n$, generates a random m-tuple $(\beta_{i1}, \dots, \beta_{im}) \in \{0,1\}^m$ and computes

$$\sigma_{ik} = \begin{pmatrix} \beta_{i1} & \dots & \beta_{im} \end{pmatrix} A + \begin{pmatrix} 0 & \overset{n}{\dots} & 0 & s_{ik} \end{pmatrix}$$

   for any $k \in \{1, \dots, n\}$; next, it sends $(\sigma_{i1}, \dots, \sigma_{in})$ to the TTP.

3. For any $i \in \{1, \dots, l\}$ and any $k \in \{1, \dots, n\}$, the TTP recovers s_{ik} to obtain a new (secret) key $s = (s_1, \dots, s_n)$, where $s_k = s_{1k} + \dots + s_{lk} \in \mathbb{Z}_q$; then, as in the first step, it produces the matrix

$$B = \begin{pmatrix} a'_{11} & \dots & a'_{1n} & b'_1 \\ a'_{21} & \dots & a'_{2n} & b'_2 \\ \vdots & \ddots & \vdots & \vdots \\ a'_{m1} & \dots & a'_{mn} & b'_m \end{pmatrix}$$

where each $b'_j = \sum_{k=1}^{n} a'_{jk} s_k + e'_j$.

4. The TTP sends the group key B to all devices and the bootstrapping key, corresponding to s, to the aggregator.

3.2 Data aggregation

The second part of the protocol consists in data aggregation: each device sends an encryption of its plaintext $\mu_i \in \mathbb{Z}_q$ to the aggregator, which obtains $\mu_1 + \ldots + \mu_l$ without any decryption process. It works as follows (see also Fig.2).

1. Each D_i chooses $\gamma_{i1}, \ldots, \gamma_{im} \in \{0,1\}$, $c_1, \ldots, c_n \in \mathbb{Z}_q$ and, using its secret key $(s_{i1}, \ldots, s_{in}) \in \mathbb{Z}_q^n$, computes

$$\left(\gamma_{i1} \ \ldots \ \gamma_{im}\right) C + \left(0 \ \overset{n}{\ldots} \ 0 \ \ c_1 s_{i1} + \ldots + c_n s_{in}\right) = \left(* \ \overset{n}{\ldots} \ * \ \ z_i\right)$$

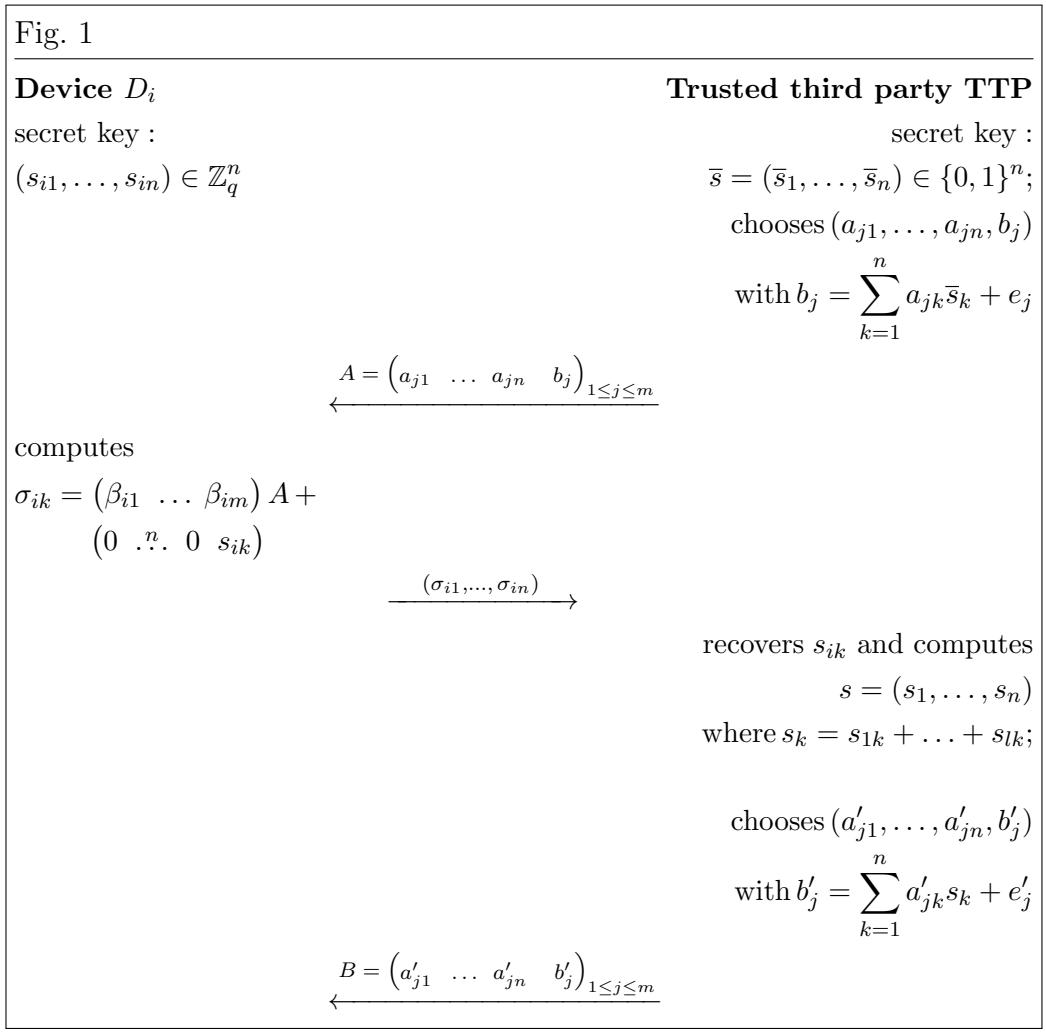
   with $C = A$ or B, as in Subsection 3.1.

2. For a given plaintext $\mu_i \in \mathbb{Z}_q$ and some $\delta_{i1}, \ldots, \delta_{im} \in \{0,1\}$, each D_i sends

$$\left(\delta_{i1} \ \ldots \ \delta_{im}\right) B + \left(0 \ \overset{n}{\ldots} \ 0 \ \ \mu_i + z_i\right) = \left(\alpha_{i1} \ \ldots \ \alpha_{in} \ \ \zeta_i\right)$$

   to the aggregator. Hence, $\zeta_i = \sum_{k=1}^{n} \alpha_{ik} s_{ik} + \mu_i + z_i + \bar{e}_i$ for some $\alpha_{i1}, \ldots, \alpha_{in} \in \mathbb{Z}_q$ and $\bar{e}_i \in \mathcal{U}$.

Fig. 1



3. The aggregator, applying the boostrapping if needed, adds up the received ciphertexts:

$$\sum_{i=1}^{l}(\alpha_{i1}, \ldots, \alpha_{in}, \zeta_i) = \sum_{i=1}^{l} E_s\big(\mu_i + z_i, \alpha_i, \bar{e}_i\big) = (a_1, \ldots, a_n, b),$$

with $\alpha_i = (\alpha_{i1}, \ldots, \alpha_{in})$; after that, it sends $(a_1, \ldots, a_n)$ to all devices.

4. Each D_i, using its own secret key $(s_{i1}, \ldots, s_{in})$, determines

$$d_i = a_1 s_{i1} + \ldots + a_n s_{in} + z_i$$

and sends it to the aggregator.

5. The aggregator gets $\mu_1 + \ldots + \mu_l$ (modulo the noise) just computing $b - (d_1 + \ldots + d_l)$.

Fig. 2

Device D_i		**Aggregator**
plaintext: $\mu_i \in \mathbb{Z}_q$;		
computes		
$\begin{pmatrix}\gamma_{i1} & \ldots & \gamma_{im}\end{pmatrix} C +$		
$\begin{pmatrix}0 & \overset{n}{\ldots} & 0 & c_1 s_{i1} + \ldots + c_n s_{in}\end{pmatrix} = \begin{pmatrix}* & \overset{n}{\ldots} & * & z_i\end{pmatrix}$		
and		
$\begin{pmatrix}\delta_{i1} & \ldots & \delta_{im}\end{pmatrix} B +$		
$\begin{pmatrix}0 & \overset{n}{\ldots} & 0 & \mu_i + z_i\end{pmatrix} = \begin{pmatrix}\alpha_{i1} & \ldots & \alpha_{in} & \zeta_i\end{pmatrix}$		
	$\xrightarrow{(\alpha_{i1},\ldots,\alpha_{in},\zeta_i)}$	
		adds up the ciphertexts:
		$\sum_{i=1}^{l}(\alpha_{i1},\ldots,\alpha_{in},\zeta_i) =$
		$(a_1,\ldots,a_n,b)$
	$\xleftarrow{(a_1,\ldots,a_n)}$	
determines		
$d_i = a_1 s_{i1} + \ldots + a_n s_{in} + z_i$		
	$\xrightarrow{d_i}$	
		gets $\mu_1 + \ldots + \mu_l$
		computing $b - (d_1 + \ldots + d_l)$

The following result guarantees the correctness of the protocol transmission.

Proposition 3.1. *At the end of the protocol, the aggregator obtains* $\mu_1 + \ldots + \mu_l$ *if all parties act honestly.*

Proof. First, recall that

$$b = \sum_{k=1}^{n} a_k s_k + \sum_{i=1}^{l}(\mu_i + z_i) + e$$

for some $e \in \mathcal{U}$, where $s_k = s_{1k} + \ldots + s_{lk}$ for any $k \in \{1, \ldots, n\}$. Taking into

account that $d_i = a_1 s_{i1} + \ldots + a_n s_{in} + z_i$ for any $i \in \{1, \ldots, l\}$, we have:

$$\begin{aligned} b - \sum_{i=1}^{l} d_i = & \\ &= \sum_{k=1}^{n} a_k s_k + \sum_{i=1}^{l} (\mu_i + z_i) + e - \sum_{i=1}^{l} \big(\sum_{k=1}^{n} a_k s_{ik} + z_i \big) \\ &= \sum_{k=1}^{n} \big(\sum_{i=1}^{l} a_k s_{ik} \big) + \sum_{i=1}^{l} \mu_i + \sum_{i=1}^{l} z_i + e - \sum_{i=1}^{l} \big(\sum_{k=1}^{n} a_k s_{ik} \big) + \sum_{i=1}^{l} z_i \\ &= \sum_{i=1}^{l} \mu_i + e. \end{aligned}$$

Thus, the aggregator gets $\sum_{i=1}^{l} \mu_i$ applying the map π, defined in Definition 2.1.3. □

3.3 Security analysis

The group key generated during the set-up protocol is the matrix

$$B = \begin{pmatrix} a'_{11} & \ldots & a'_{1n} & b'_1 \\ a'_{21} & \ldots & a'_{2n} & b'_2 \\ \vdots & \ddots & \vdots & \vdots \\ a'_{m1} & \ldots & a'_{mn} & b'_m \end{pmatrix}$$

where $b'_j = \sum_{k=1}^{n} a'_{jk} s_k + e'_j$ and each s_k is only known to the TTP. As a consequence, obtaining $s = (s_1, \ldots, s_n)$ is hard as solving the LWE problem.

For data aggregation, each device sends first $\zeta_i = \sum_{k=1}^{n} \alpha_{ik} s_{ik} + \mu_i + z_i + \bar{e}_i$ and then $d_i = a_1 s_{i1} + \ldots + a_n s_{in} + z_i$ to the aggregator. Recall that

$$\big(\gamma_{i1} \ \ldots \ \gamma_{im}\big)\, C + \big(0 \ \overset{n}{\ldots} \ 0 \ \ c_1 s_{i1} + \ldots + c_n s_{in}\big) = \big(* \ \overset{n}{\ldots} \ * \ \ z_i\big)$$

with $\gamma_{i1}, \ldots, \gamma_{im} \in \{0, 1\}$, $c_1, \ldots, c_n \in \mathbb{Z}_q$, and $C = A$ or B as in Subsection 3.1. Then any d_i contains the noise of z_i, and the coefficient of each s_{ik} changes according to c_k. Hence, for any device, given a list of ciphertexts of the form ζ_i or d_i, recovering its secret key $(s_{i1}, \ldots, s_{in})$ is very difficult assuming that the LWE problem is hard.

Note finally that if one device is corrupted, the privacy of others is still guaranteed but we cannot obtain just the aggregation of all plaintexts of the

honest devices. Moreover, if a device is eliminated or a new one is added, we need to perform the group key phase again. This makes our solution more suitable for scenarios where the set of devices is relatively static.

References

[1] J.-M. Bohli, C. Sorge and O. Ugus, *A Privacy Model for Smart Metering*, Proceedings of the First IEEE International Workshop on Smart Gride Communications (in conjunction with IEEE ICC 2010), 2010.

[2] Z. Brakerski, A. Langlois, C. Peikert, O. Regev and D. Stehlé, *Classical hardness of learning with errors (extended abstract)*, STOC'13– Proceedings of the 2013 ACM Symposium on Theory of Computing, pages 575–584.

[3] N. Busom, R. Petrlic, F. Sebé, C. Sorge and M. Valls, *Efficient smart metering based on homomorphic encryption*, Computer Communications, **82** (2016), pages 95–101.

[4] I. Chillotti, N. Gama, M. Georgieva and M. Izabachene, *TFHE: fast fully homomorphic encryption over the torus*, J. Cryptology **33** (2020), no. 1, pages 34–91.

[5] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila and S. Tap, *CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE*, WAHC 2020 – 8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Dec 2020, [Virtual], France.

[6] R. Cramer, R. Gennaro and B. Schoenmakers, *A Secure and Optimally Efficient Multi-Authority Election Scheme*, European Transactions on Telecommunications, **8**(5) (1997), pages. 481–490.

[7] L. Ducas and D. Micciancio, *FHEW: Bootstrapping homomorphic encryption in less than a second*, In: Oswald, E., Fischlin, M. (eds) Advances in Cryptology – EUROCRYPT 2015, Part I, 617–640, Lecture Notes in Comput. Sci., 9056, Springer, Heidelberg, 2015.

[8] M. Ferrara, A. Tortora and M. Tota, *An overview of torus fully homomorphic encryption*, Int. J. Group Theory (2023), doi: 10.22108/IJGT.2023.139030.1869.

[9] S. Finster and I. Baumgart, *Privacy-Aware Smart Metering: A Survey*. IEEE Communications Surveys & Tutorials, vol. 17 (2015), no. 2, pp. 1088-1101.

[10] C. Gentry, *Computing arbitrary functions on encrypted data*, Communications of the ACM **53** (2010), no. 3, 97–105.

[11] C. Gentry, *Fully homomorphic encryption using ideal lattices*, STOC'09–Proceedings of the 2009 ACM International Symposium on Theory of Computing, 169–178, Association for Computing Machinery, New York, 2009.

[12] M. Jawurek, F. Kerschbaum and G. Danezis *Sok: Privacy Technologies for Smart Grids - A Survey of Options*, Technical report, Microsoft Technical Report, 2012.

[13] V. Lyubashevsky, C. Peikert and O. Regev, *On ideal lattices and learning with errors over rings*. In: Gilbert, H. (eds) Advances in Cryptology – EUROCRYPT 2010. EUROCRYPT 2010. Lecture Notes in Computer Science, vol 6110. Springer, Berlin, Heidelberg, pages 1–23, 2010.

[14] P. Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: Stern, J., editor, Advances in Cryptology – EUROCRYPT '99. EUROCRYPT 1999. Lecture Notes in Computer Science, vol.1592, pages 223–238, 1999.

[15] C. Peikert, *Public-key cryptosystems from the worst-case shortest vector problem: extended abstract*. STOC'09–Proceedings of the 2009 ACM International Symposium on Theory of Computing, 333–342. Association for Computing Machinery (ACM), New York, 2009.

[16] O. Regev, *Lattice-based cryptography*. In: Dwork, C. (eds), Advances in Cryptology - CRYPTO 2006. Lecture Notes in Computer Science, vol 4117, pages 131–141. Springer, Berlin, Heidelberg.

[17] O. Regev, *On lattices, learning with errors, random linear codes, and cryptography*, Journal of the ACM **56** (2009), no. 6, Art. 34, 40 pp.

[18] R. Rothblum, *Homomorphic encryption: From private-key to public-key*. In Y. Ishai, editor, Theory of Cryptography (TCC 2011), volume 6597 of Lecture Notes in Computer Science, pages 219–234. Springer, Berlin, Heidelberg, 2011.